NASREM AS A FUNCTIONAL ARCHITECTURE FOR THE DESIGN OF ROBOT CONTROL SYSTEMS

DR. RONALD LUMIA
ROBOT SYSTEMS DIVISION
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

ABSTRACT

A functional architecture is defined to be a technology independent specification of a system while a computer architecture is a technology dependent realization of a functional architecture. The functional architecture serves as the "source" from which all implementations may be traced. This paper reviews several functional architectures available in the literature for robot control systems. For this paper, the term robot control system refers to a complete system which includes the sensory and modeling components of the system as well as the actual control loops. The NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM) is presented along with the NIST approach to its realization. Interfaces for the functional architecture are defined using the literature so that extant algorithms can be implemented and evaluated in the system. This approach is illustrated by considering the SERVO level of NASREM in detail. Then, one possible computer architecture for the realization of NASREM is shown. The result of choosing the NASREM functional architecture is that it provides a technology independent paradigm which can be mapped into a technology dependent implementation capable of evolving with technology in the laboratory as well as in space.

1. INTRODUCTION

System designers tend to approach problems with specific hardware and software biases. Given some specific requirements, the approach should be a careful analysis of available hardware and software. However, because of economic as well as other constraints, the process often degenerates into rationalizing why a favorite computer language and familiar computer chips are precisely what is need to solve the problem. Often, the requirements of the problem are manipulated into the form of how to solve the problem using blackboards, whiteboards, hierarchical control, heterarchical control, etc. The approach that is chosen ultimately depends upon the research organization, project manager, or thesis advisor. It would be illuminating to try to forget some of these biases for a moment and instead concentrate on the real requirements of the system. This paper will, at

least initially, try to refrain from the institutional bias of the solution in order to define the problem better.

One approach to the design of a system is to analyze the problem in order to develop a sufficient set of requirements that the problem itself imposes upon the solution. Then these requirements drive the solution. This approach, often called a "point design" provides a system that does exactly (hopefully) what it is supposed to do. Unless the specifications for the system are written with extraordinary care, however, it is often the case that the system must be modified to satisfy some previously unstated but crucial requirements. Unfortunately, it is often very difficult to modify a point design because the system was designed to satisfy the design requirements efficiently. It was not designed to be flexible. Consequently, the tradeoff between efficiency and flexibility is critical in obtaining a useful system.

Another approach to the design of a system is to develop a generic functional architecture which encompasses all of the functions associated with the ideal system. In this approach, efficiency may be sacrificed for flexibility. However, if the capabilities of the system are subject to change over time, then this flexibility is imperative. The assumption of this paper is that there is no clearly superior approach to the design of robot control systems and therefore opting for flexibility is the most reasonable approach.

The word "architecture" has been used extensively in the literature but a precise definition does not currently exist. In this paper, there is a conscious attempt to separate the concept of the functional architecture from that of the computer architecture. As its name suggests, the functional architecture is concerned with the functional aspects of the problem and every attempt is made to make it technology independent. On the other hand, the computer architecture is clearly technology dependent. It represents a particular realization of the functional architecture at some point in time. Consequently, if the same functional architecture were implemented at a different time, it is not unreasonable to expect that a totally different computer architecture would be used in order to take advantage of improvements in the state-of-the-art.

Given these definitions, the design procedure can be summed up in the following steps:

1. Development of a functional architecture.

2. Development of a set of interfaces for the functional architecture.

3. Development of the computer architecture.

4. Software and hardware development, testing, and

integration.

In looking through the literature, it is evident that many researchers have developed architectures for robot control. Very few of these architectures, however, are functional architectures as previously defined because they are technology dependent. Because of the immense amount of experience many researchers possess, the design procedure often begins in the second or third phase of the design process. Familiarity with the problem seems to lead researchers to solve the problem without first formally defining the functional architecture. As a result, the realizations are far less flexible than they might be. This process is essentially a "point design." Each researcher begins the design process knowing precisely which algorithms are best and an efficient system to execute these algorithms is developed. There are no systems with the capability to easily interface other control algorithms, robot models, sensory processors, etc., without major system redesign. Consequently, looking seriously into the development of a functional architecture could have a profound effect on the research community because of the possibility of comparing competing approaches.

This paper is organized in the following way. The next section will review some of the functional architectures associated with complete robot control systems. This will be followed by a description of NASREM. The approach to the NIST implementation of NASREM will be presented.

2. ALTERNATIVE FUNCTIONAL ARCHITECTURES

The vast majority of work in robot control systems would not be cited as functional architectures because of technology dependence. Several functional architectures exist and this section will described three of them.

Saridis [1] develops the concept of "intelligent control" for his functional architecture. Intelligent control is the marriage of artificial intelligence, control and operations research. The approach employs hierarchy centered around the guiding principle that there is decreasing precision with increasing intelligence. Lower levels in the hierarchy are fast and precise but fundamentally dumb. For example, in the control of a robot, the level which controls the joints of the robot must support a certain update rate for performance and stability. However, it does not need to deal with the planning for why the action is actually being performed. The functional architecture specifies three levels in the hierarchy which in descending order of intelligence are: the organization level, the coordination level, and the hardware control level.

Brooks [2] describes a functional architecture based on what he calls the subsumption principle. While the approach is applied

to mobile robots, it is clearly more generic. The basic idea is that there is a hierarchy of functional modules which communicate over low bandwidth channels. Each level in the hierarchy is responsible for a certain function. For example, one level is responsible for the goal of wandering around without colliding with objects in the environment. A level superior to this could take over some of the functions of the inferior level or "subsume" its functionality by suppressing the lower level's outputs. As a result, multiple goals, multiple sensors, robustness, and extensibility are possible when this functional architecture is implemented. Brooks claims that subsumption is superior to task decomposition because a level can perform functions without instruction from a superior level. However, this objection may be less a complaint about task decomposition than about particular implementations. The subsumption architecture itself has task decomposition in that each functional module in the hierarchy is, in effect, decomposing a task into the functions appropriate for the next level. The objection Brooks may be describing is a system where all planning is done in one box, all control in another, etc.

The subsumption architecture does exhibit some potential problems. All sensory processing information is sent to all levels and it is up to each level to pay attention or ignore the data as it sees fit. Consequently, certain processes which are inherently sequential may cause some problems. For example, in processing an image, the identification and location of an object through a noise reduction, feature extraction, and feature classification sequence need not communicate with all layers in the hierarchy since it is clear that certain levels will definitely have no use whatsoever for the data. A more serious concern about the subsumption architecture is that each level of competence includes as a subset each earlier level of competence. The problem is that it may be very difficult for the upper level to take into account all of the details that the lower level must deal with in order to be able to subsume its functionality. Interestingly, this seems to be contrary to Saridis' approach where intelligence and precision at hierarchical levels are inversely related. The concept of hierarchy is normally used precisely to avoid having higher levels know the details of lower level's processes.

Shafer [3] presents a functional architecture for sensory perception called CODGER which is based to some extent on the Hearsay [4] system. In his system it is anticipated that the perception subsystem is expected to use 90 % of the compute power. There is a heterarchy consisting of a central database, pool of knowledge intensive modules, and database manager which synchronizes the modules. The approach is presented in this section even though it deals primarily with the sensory processing aspects of the problem because there is a conscious attempt to define a functional architecture before proceeding with the design. The system philosophy is to provide as much top-down guidance as possible and to exploit sensor modality

difference to produce complementary rather than competing perceptual processes. CODGER is a communications database with geometric reasoning. It has a whiteboard which is equivalent to a blackboard with parallel execution of modules including geometric reasoning. CODGER is used for the Autonomous Land Vehicle (ALV) where the processing associated with the perception of the environment is the bottleneck. The other parts of the ALV, the captain for example, are relatively easy because a human does nearly all of the complex planning.

3. NASA/NBS STANDARD REFERENCE MODEL FOR TELEROBOT CONTROL SYSTEM ARCHITECTURE (NASREM)

Another functional architecture is the NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM). While NASREM focuses on the application of robots to space, the same approach has been used to develop robot systems for manufacturing [5], undersea vehicles [6], remotely driven vehicles [7], shipbuilding robots [8], etc., and it is therefore argued that the architecture is, in some sense, generic for robot control systems. The fundamental paradigm of the control system is shown in Figure 1. The control system architecture is a three legged hierarchy of computing modules, serviced by a communications system and a global memory. The task decomposition modules perform real-time planning and task monitoring functions; they decompose task goals both spatially and temporally. The sensory processing modules filter, correlate, detect, and integrate sensory information over both space and time in order to recognize and measure patterns, features, objects, events, and relationships in the external world. The world modeling modules answer queries, make predictions, and compute evaluation functions on the state space defined by the information stored in global memory. Global memory is a database which contains the system's best estimate of the state of the external world. The world modeling modules keep the global memory database current and consistent.

The first leg of the hierarchy consists of task decomposition modules which plan and execute the decomposition of high level goals into low level actions. Task decomposition involves both a temporal decomposition (into sequential actions along the time line) and a spatial decomposition (into concurrent actions by different subsystems). Each task decomposition module at each level of the hierarchy consists of a job assignment manager, a set of planners, and a set of executors. These decompose the input task into both spatially and temporally distinct subtasks.

The second leg of the hierarchy consists of world modeling modules which model (i.e., remember, estimate, predict) and evaluate the state of the world. The "world model" is the system's best estimate and evaluation of the history, current state, and possible future states of the world, including the

states of the system being controlled. The "world model" includes both the world modeling modules and a knowledge base stored in a global memory database where state variables, maps, lists of objects and events, and attributes of objects and events are maintained. The world model maintains the global memory knowledge base by accepting information from the sensory system, provides predictions of expected sensory input to the corresponding sensory system modules, based on the state of the task and estimates of the external world, answers "What is?" questions asked by the executors in the corresponding task decomposition modules, and answers "What if?" questions asked by the planners in the corresponding task decomposition modules.

The third leg of the hierarchy consists of sensory processing sensory system modules. These recognize patterns, detect events, and filter and integrate sensory information over space and time. The sensory system modules at each level compare world model predictions with sensory observations and compute correlation and difference functions. These are integrated over time and space so as to fuse sensory information from multiple sources over extended time intervals. Newly detected or recognized events, objects, and relationships are entered by the world modeling modules into the world model global memory database, and objects or relationships perceived to no longer exist are removed. The sensory system modules also contain functions which can compute confidence factors and probabilities of recognized events, and statistical estimates of stochastic state variable values.

The control architecture has an operator interface at each level in the hierarchy. The operator interface provides a means by which human operators, either in the space station or on the ground, can observe and supervise the telerobot. Each level of the task decomposition hierarchy provides an interface where the human operator can assume control. The task commands into any level can be derived either from the higher level task decomposition module, from the operator interface, or from some combination of the two. Using a variety of input devices, a human operator can enter the control hierarchy at any level, at any time of his choosing, to monitor a process, to insert information, to interrupt automatic operation and take control of the task being performed, or to apply human intelligence to sensory processing or world modeling functions.

The sharing of command input between human and autonomous control need not be all or none. It is possible in many cases for the human and the automatic controllers to share control of a telerobot system simultaneously. For example, in an assembly operation, a human might control the position of an end effector while the robot automatically controls its orientation. For a more detailed description of NASREM, see [9].

4. NIST IMPLEMENTATION OF NASREM

In order to implement a functional architecture, especially one like NASREM which allows evolution with technology, the interfaces must be carefully defined. Although the NASREM functional architecture specifies the purpose of each module in the control system hierarchy, it does not completely specify the interfaces between modules. This section will describe the method by which the interfaces for the SERVO level of the hierarchy have been defined. The method involves gathering all of the algorithms available for SERVO level control, dividing each algorithm into the parts which inherently belong to task decomposition, world modeling, and sensory processing, and then deriving the interfaces which will support these algorithms. Any design, however, must constrain the problem sufficiently so that detailed interfaces can be devised.

With this in mind, the Servo Level design was based on a fundamental control approach which computes a motor command as a function of feedback system state y, desired state (attractor) y_d, and control gains. In this approach, the gains are coefficients of a linear combination of state errors $(y-y_d)$. The system state and its attractor are composed from the physical quantities to be controlled, (i.e., position, force, etc.,) and can be expressed in an arbitrary coordinate system. This type of algorithm is the basis for almost all manipulator control schemes [10]. However, this basic algorithm is inadequate for controlling the gross aspects of manipulator motion, as described in [11]. The algorithm can provide "small" motions so that the dynamics of the servo algorithm itself are not significant. This means that the Primitive Level must generate the gross dynamics of the motion through a sequence of inputs to the Servo Level. This can be achieved through an appropriate sequence of either attractor points [10,12] or gain values [11].

Figure 2 depicts the detailed Servo Level design. The task decomposition module at the Servo Level receives input from Primitive in the form of the command specification parameters. The command parameters include a coordinate system specification C_z which indicates the coordinate system in which the current command is to be executed. C_z can specify joint, end-effector, or Cartesian (world) coordinates. Given with respect to this coordinate system are desired position, velocity, and acceleration vectors $(z_d,\ z_d,\ z_d)$ for the manipulator, and the desired force and rate of change of force vectors $(f_d,\ f_d)$. These command vectors form the attractor set for the manipulator. The K's are the gain coefficient matrices for error terms in the control equations. The selection matrices (S,S') apply to certain hybrid force/position control algorithms. Finally, the "Algorithm" specifier selects the control algorithm to be executed by the Servo Level.

When the Servo Level planner receives a new command specification, the planner transmits certain information to world

modeling. This information includes an attention function which tells world modeling where to concentrate its efforts, i.e. what information to compute for the executor. The executor simply executes the algorithm indicated in the command specification, using data supplied by world modeling as needed.

The world modeling module at the Servo Level computes model-based quantities for the executor, such as Jacobians, inertia matrices, gravity compensations, Coriolis and centrifugal force compensations, and potential field (obstacle) compensations. In addition, world modeling provides its best guess of the state of the manipulator in terms of positions, velocities, end-effector forces and joint torques. To do this, the module may have to resolve conflicts between sensor data, such as between joint position and Cartesian position sensors.

Sensory processing, as shown in Figure 2, reads sensors relevant to Servo and provides the filtered sensor readings to world modeling. In addition, certain information is transmitted up to the Primitive Level of the sensory processing hierarchy. Primitive uses this information, as well as information from Servo Level world modeling, to monitor execution of its trajectory. Based on this data, Primitive computes the stiffness (gains) of the control, or switches control algorithms altogether. For example, when Primitive detects a contact with a surface, it may switch Servo to a control algorithm that accommodates contact forces.

A more complete description of the Servo Level is available in [10] where the vast majority of the existing algorithms in the literature are described. The same process for developing the interfaces based on the literature has also been performed for the Primitive level and is available in [12]. While the procedure is planned for each level in the hierarchy, the amount of literature support tends to decrease as one moves up the hierarchy.

Once the interfaces are defined, it is possible to choose a computer architecture and begin to realize the system. While every effort is being made to do the job properly, there is no reason to assume that the implementation at NIST is optimal in any way. It is simply illustrates one realistic method to implement the NASREM architecture.

While a functional architecture is technology independent, its implementation obviously depends entirely on the state-of-the-art of technology. The designer must choose existing computers, buses, languages, etc., and, from these tools, produce a computer architecture capable of performing the functions of the functional architecture. The system must adequately meet the real-time aspects of the controller so that adequate performance is achieved through careful consideration of computer choice, multiple processor real-time operating system, inter-processing communication requirements, tasking within certain processors,

etc. For a more detailed description of this methodology, see [13].

The NIST implementation considers two aspects of the process: the development environment on which the code is written, debugged, and tested as well as possible, and the target environment where the code for the real-time robot control system is executed. Figure 3 shows the approach. A network of SUN workstations running UNIX is used for the development environment, sacrificing the speed of the developed code for the ease of development. Once the code is tested as well as possible, it is downloaded to the target system. The target system consists of a VME backplane of several (currently 6) 68020 processors. For rapid iconic image processing, the PIPE system [14] is integrated into the system. The target hardware drives a Robotics Research Corp. K-1607 arm.

From the software side, the multiprocessing operating system used for the target is required to be as simple as possible so that the overhead is minimized. The duties of the operating system are limited to very simple actions such as downloading executable code, starting up the processors, and interprocessor communication. While tasking is not performed at the lower levels of the hierarchy because of the overhead associated with context switches, it is desirable at higher levels in the hierarchy which are not as time critical. NIST researchers are currently investigating three alternatives for tasking: tasking provided by the native ADA compiler, pSOS tasking, and ADA tasking. Interprocessor communications alternatives including pRISM, sockets, etc., must also be evaluated empirically. The actual application code is written in ADA. Although ADA compilers cannot currently produce code as efficient as other languages such as C, NIST researchers have shown that the gap is steadily decreasing [15].

The application code is developed by programming the processes which achieve the functions associated with the boxes in the functional architecture. The problem then becomes one of assigning each of the processes, such as those shown in Figure 2, to a particular processor. There is a clear trade-off between the cost of the solution and the performance of the system. There are currently no software tools which automatically perform this assignment based on an arbitrary index of performance. The approach at NIST is step-wise refinement of the performance of the system. Given the particular hardware being used, a certain number of processors is chosen arbitrarily. For that configuration, the processes are assigned to the processors. Then, the system is evaluated in terms of its performance. If the performance is unacceptable, the designer has several options. The first option is to add more processors. This alternative is balanced against additional communication required by the processors. Another alternative is to add faster processors or special purpose processors, such as dynamics chips, which optimize particularly compute intensive operations. This

trade-off clearly relates to cost. Another alternative is to reassign the processes to the processors in order to balance the workload of each processor. Each of the alternatives can be used by the designer in order to improve the performance of the system. This allows a particular configuration which implements the functional architecture to change with time as improvements in technology are realized.

5. CONCLUSION

While there are many competing computer architectures for robot control, there are relatively few functional architectures. Some purport to be general in nature while others are satisfied with achieving a set of specifications for a particular application. The ideal situation would be if there were some way to prove mathematically which architecture is the best. Unfortunately, this is does not appear to be possible at the present time because the proof of optimality is intimately connected with defining and understanding intelligence. If it is difficult to define intelligence, it should be no wonder that building an intelligent system has also been shown to be a formidable task.

Being unable to prove optimality cannot imply that research in the area of functional architectures for robot control is futile. Architectures which have been considered to be generic have been suggested but there is really no way to discuss the relative merits of each approach without empirically obtained evidence. The most prudent course of action would be to implement some of these "general" architectures and find out precisely what is right and wrong with each. Hopefully, this approach, which can be considered to be step-wise refinement, will lead to more global generalizations of the robot control problem, and ultimately suggest an architecture which holds the paradigm for truly intelligent robot behavior in a machine.

ACKNOWLEDGMENTS

The author would like to thank NASA's Goddard Space Flight Center for sponsoring this work under contract S-28187-D.

REFERENCES

[1] G.N. Saridis, "Foundations of the Theory of Intelligent Controls," IEEE Workshop on Intelligent Control, Troy, August, 1985, p. 23.

[2] R. Brooks, "A Robust Layered Control System for a Mobile Robot," IEEE Journal of Robotics and Automation, Vol 2,

number 1, 1986, p. 14.
[3] S.A. Shafer, A. Stentz, C. Thorpe, "Vision and Navigation for the Carnegie-Mellon Navlab", IEEE Int. Conf. Robotics and Automation, San Francisco, April, 1986 p. 2002.
[4] V.R.Lesser, R.D. Fennell, L.D. Erman, D.R. Reddy, "Organization of the Hearsay II Speech Understanding System," IEEE Trans. on ASSP, Vol. ASSP-23, number 1, 1975, p. 11.
[5] J.A. Simpson, R.J. Hocken, J.S. Albus. "The Automated Manufacturing Research Facility of the National Bureau of Standards," Journal of Manufacturing Systems, 1, 1, 1982, 17.
[6] J.S. Albus, "System Description and Design Architecture for Multiple Autonomous Undersea Vehicles," NIST Technical Note 1251, September, 1988.
[7] S. Szabo, H.A. Scott, R.D. Kilmer, "Control System Architecture for the TEAM Program," Proc. 2nd Int. Symp on Robotics and Manufacturing, Albuquerque, New Mexico, November, 1988, p. 483.
[8] N.G. Dagalakis, J.S. Albus, B.L. Wang, J. Unger, J.D. Lee, "Stiffness Study of a Parallel Link Robot Crane for Shipbuilding Applications," Proc. 7th Int. Conf. on Offshore Mechanics and ARctic Engineering, Vol. VI, Houston, Texas Feb., 1988, p. 29.
[9] J.S. Albus, R. Lumia, H.G. McCain, "NASA/NBS Standard Reference Model For Telerobot Control System Architecture (NASREM)," NBS Technote #1235, also as NASA document SS-GSFC-0027.
[10] J. Fiala, "Manipulator Servo Level Task Decomposition," NIST Technote #1255, April 20, 1988.
[11] J. Fiala, "Generation of Smooth Trajectories without Planning," manuscript in preparation.
[12] A.J. Wavering, "Manipulator Primitive Level Task Decomposition," NIST Technote #1256, January 5, 1988.
[13] T. Wheatley, J. Michaloski, and R. Lumia, "Requirements for Implementing Real Time Control Functional Modules on a Hierarchical Parallel Pipelined System," 1989 Conference on Space Telerobotics, Pasadena, January 30, 1989.
[14] E.W. Kent, M.O. Shneier, and R. Lumia, "PIPE," Journal of Parallel and Distributed Computing, Vol. 2, 1985, pp. 50-78.
[15] S. Leake, "A Comparison of Robot Kinematics in ADA and C on Sun and microVAX," Robotics and Automation Session, IASTED, Santa Barbara, CA., May 25-27,1988.

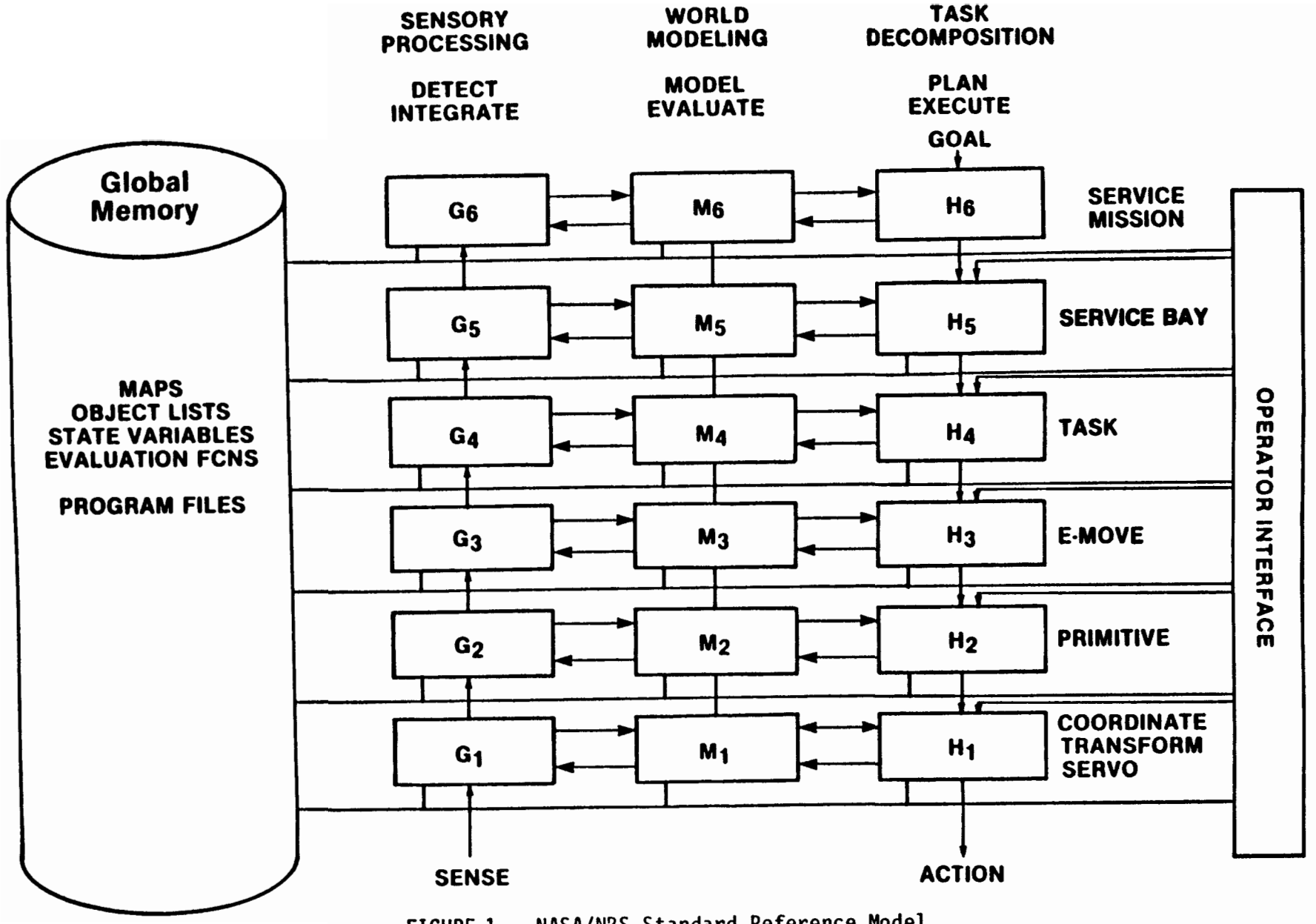

FIGURE 1 - NASA/NBS Standard Reference Model
for Telerobot Control System Architecture (NASREM)

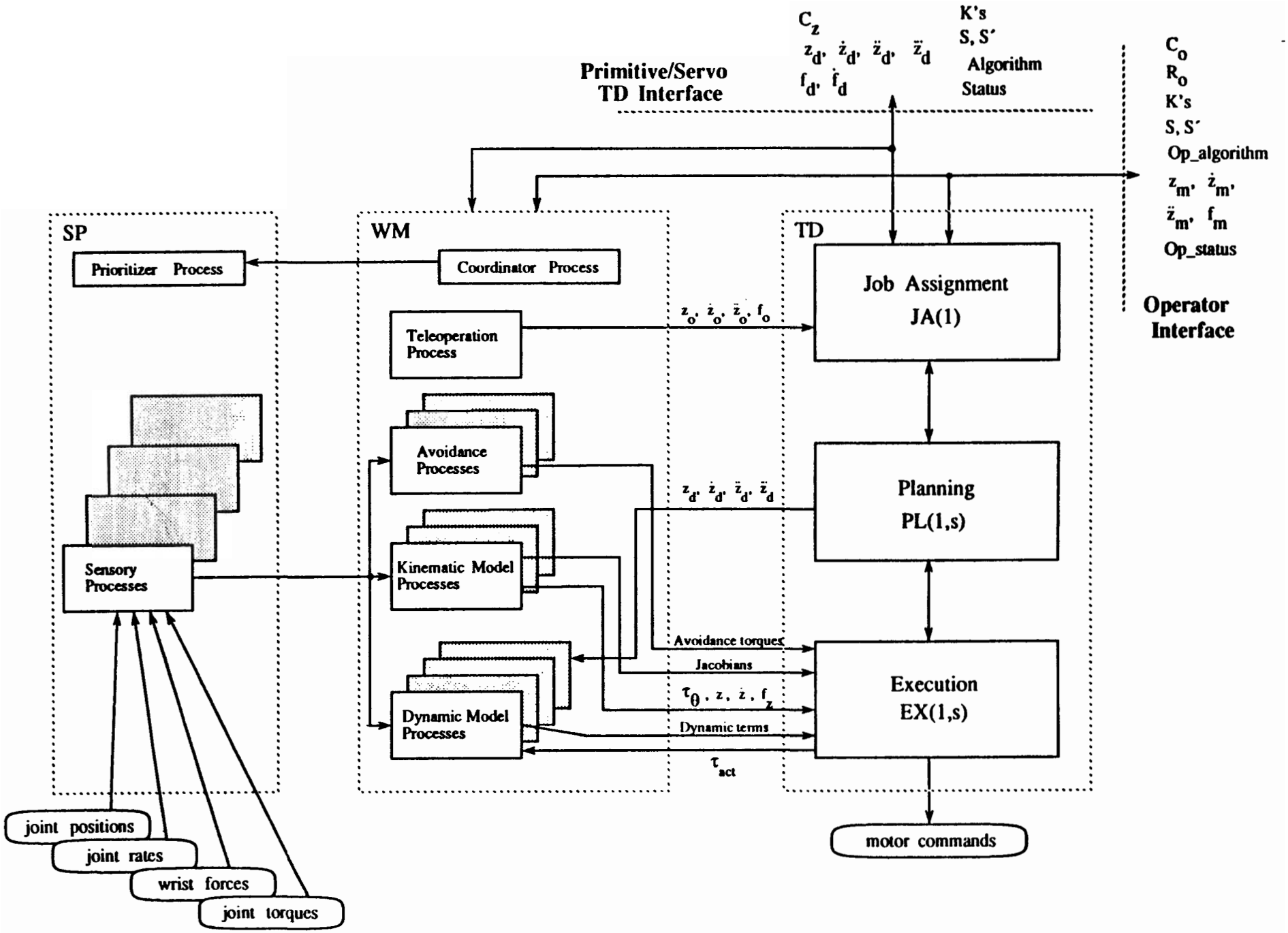

FIGURE 2 - Servo Level Interfaces

SYSTEM DEVELOPMENT
(View at Hardware)

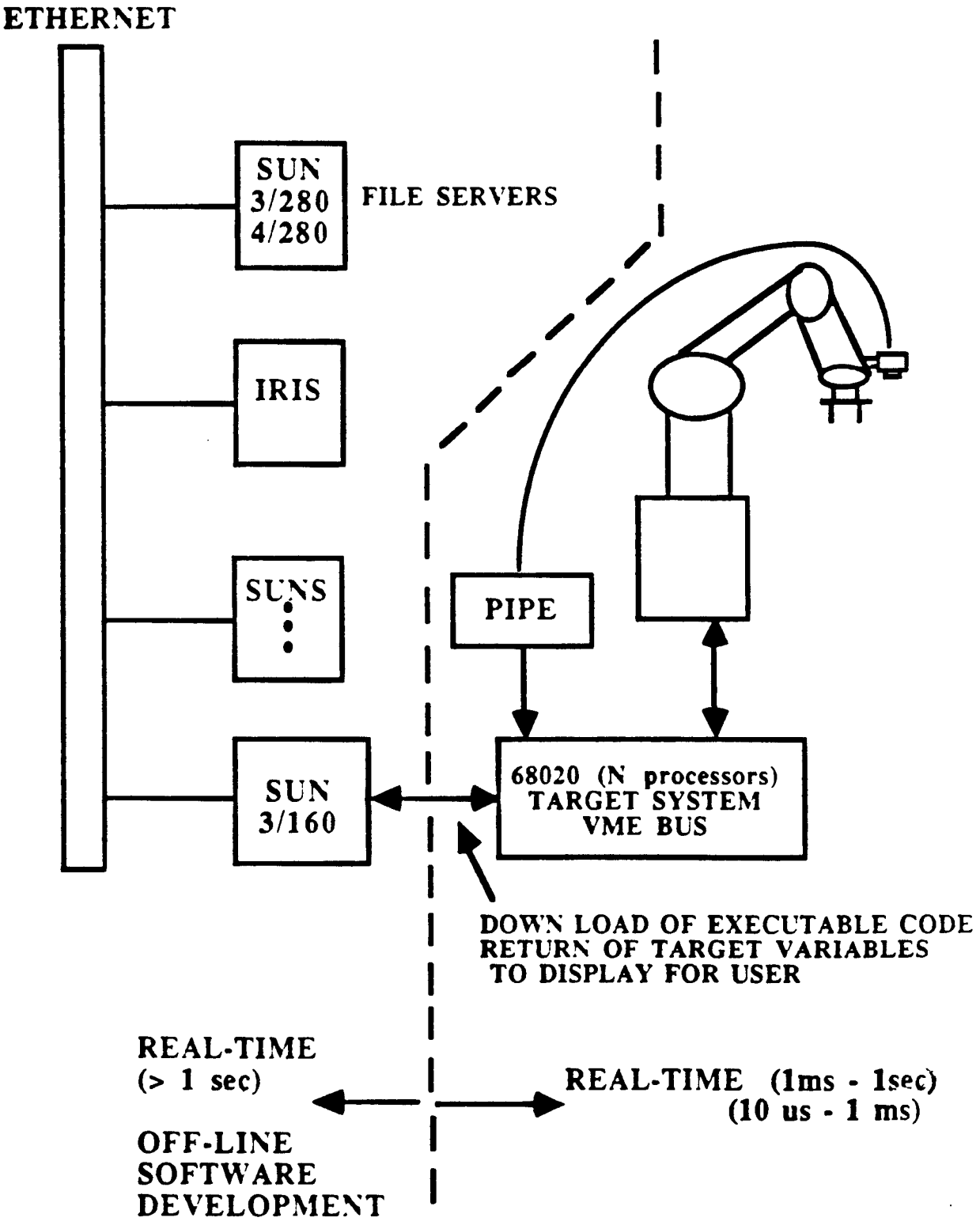

FIGURE 3 - NIST Implementation of NASREM